Non-regular Languages

Reading

Section 1.4

Practice problems (page 85): 1.29, 1.30, 1.47, 1.48, 1.49

Challenge: 1.44, 1.45, 1.54, 1.55, 1.56, 1.63

Non-regular Languages

It is important to understand the limitations of any system we use, and this includes finite automata. So a natural question there is: Can we easily say if a language is regular or not?

> A language is called **non-regular**, if there is no DFA/NFA that recognizes it.

So how can we tell if a language is non-regular? We can show a language is regular by providing a DFA/NFA for it, but we cannot say a language is non-regular simply because we have not been able to find a DFA for it.

> Our inability to find a proof for a statement is not a proof that the statement is false.

We therefore need something more concrete. But before we do that, let us state a question that we will return to at a later time:

> Is there a process/algorithm/method, which given a language can determine if that language is regular or non-regular?

Let's attempt a partial answer:

- Can we perhaps go through each DFA out there, in some order (we can probably order them in some way starting by size) and try each DFA to see if it matches the language?
  - First problem: If the language is not finite, we don't really even have a way to make sure that the DFA matches it.
  - Second problem: If the language is non-regular, we would have to check every DFA in existence, and there are infinitely many of them. So we would never terminate.

Let us leave it at that for now.

But let us exemplify some of our difficulties with the following two examples:

The language

$$C = \{w \mid w \text{ has equal number of 0s and 1s }\}$$

is not regular. But the language

$$D = \{w \mid w \text{ has equal number of 01s and 10s }\}$$

in fact is regular.

Take a moment to ponder why that is.

The Pumping Lemma for regular languages

In this section we will establish a nice relatively easy-to-test requirement for a language to be regular. This would allow us to exhibit many languages as non-regular. It is known as the *pumping lemma*. It essentially says that if the language has a string of a certain length, then we can produce infinitely many strings of a certain form that must also be in the language.

In this section we will make extensive use of the notation $|x|$ to denote the length of the string x.

> **Pumping Lemma** for regular languages.
>
> If A is a regular language, then there is a number p, called the **pumping length**, so that if s is any string in A with length at least p, then we can break s in three parts as
>
> $$s = xyz$$
>
> so that:
>
> 1. for each $i \geq 0$ we have that $xy^iz \in A$,
> 2. $|y| > 0$ (y is nonempty),
> 3. $|xy| \leq p$.

Here's the idea of the proof:

- Since A is a regular langauge, there is some DFA that recognizes the language. Let's say that it has p states.
- We start on state q_0, and as we trace the string s through the DFA we must pass through at least $p + 1$ states (including the start state).
- But there are only p distinct states, so it must be the case that as we trace the string s through the DFA, it passes through the same state at two distinct instances.

- Looking at the points at which this occurs is what allows us to break the string up into the three parts xyz: x gets us to that repeated state for the first time, y does a loop and takes us back to that state, then z continues to the end.
- Then we can use any number of ys, and they would just amount to us doing more loops at that state, or no loop at all. In any case we can continue with the z part to arrive at a final state. So all the strings xy^iz are in the language.

Now let us look at a formal proof:

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes the language A, and p be the size of Q.
- Let $s = s_1 s_2 \cdots s_n$ be a string in A, with $n \geq p$.
- Since M recognizes the string, there are states r_0, r_1, ..., r_n such that:
  - $r_0 = q_0$,
  - $r_{i+1} = \delta(r_i, s_{i+1})$ for all $i = 0, \ldots, n-1$,
  - $r_n \in F$.
- There are $n+1$ of the r_i's, and only p distinct states, so some 2 of the r_i must be the same. Let us say the first one of these is r_j, and the next one is r_l (there might be more, we pick the last). Note that $l \leq p+1$.
- Let $x = s_1 s_2 \cdots s_{j-1}$, $y = s_j \cdots s_{l-1}$, $z = s_l \cdots s_n$. Then of course $w = xyz$.
- We observe that xz would also be accepted by the automaton, using the states $r_0, \ldots, r_{j-1}, r_j = r_l, r_{l+1}, \ldots, r_n$.
- xy^2z would be accepted, using the states $r_0, \ldots, r_j, \ldots, r_l = r_j, r_{j+1}, \ldots, r_l, r_{l+1}, \ldots, r_n$.
- Similarly the strings xy^iz would be accepted, by repeating the strings $r_{j+1}, \ldots, r_l$ an appropriate number of times.
- It is clear that $|y| > 0$ and that $|xy| = l - 1 \leq p$.

Question: Can it happen that x or z is empty? Can they both be empty?

Using the pumping lemma

Let us see how the pumping lemma can help us show that a language is non-regular. As an example we will take the language $A = \{0^n 1^n \mid n \geq 0\}$ described earlier. The idea applies in many situations:

1. Suppose A was regular. Then it would have a pumping length p.

2. Consider an appropriate string from the language. In this case we will consider the string $s = 0^p 1^p$.

3. Apply the pumping lemma: This string s can be written as $s = xyz$ with the aforementioned conditions.

4. The condition $|xy| \leq p$ can help us say something about y. In this case y must consist of all zeros.

5. The fact that the strings $xy^i z$ must all be in the language usually leads to a contradiction now. In our case, $xy^2 z$ has more zeros than xyz, but the same number of ones. That is not possible given that xy^z is supposed to also be in A.

6. The fact that we arrived at a contradiction tells us that the language A must not be regular.

Note that the exact same proof can be used also for the language $C = \{w \mid w$ has equal number of

The book has a lot of excellent examples. Study them!